Case Mapper

User Manual

August 2013 Beta

3rd Analogy Conference Edition

Kenneth D. Forbus, Scott Friedman, Subu Kandaswamy, Andrew Lovett, and Jeffrey Usher

Qualitative Reasoning Group

Northwestern University

Contents

Overview 4
About Knowledge Bases and Cases 4
How to: Open a Knowledge Base 5
How to: Back up a knowledge base 6
How to: Close Case Mapper 6
How to: Browse a Knowledge Base 6
How to: Fetch a Case 8
Using SME from within Case Mapper 10
How to: Run SME 10
How to: Browse SME mappings 10
How to: Constrain SME 11
Using MAC/FAC from within Case Mapper 13
How to: Select a Case Library 13
How to: Run MAC/FAC 13
Extending a knowledge base 15
How to: Write a case file 15
How to: Define new predicates and concepts 16
How to: Load a case file into your knowledge base 17
How to: Test a case file 17
KQML Server 17
How to: Start & Stop the KQML Server 18
Basic KQML & Packet Structure 18
How to: Use KQML to Ask 18
How to: Use KQML to Store 19
How to: Use KQML to Retrieve 19
How to: Use KQML to Forget 19
Using the test-client.lsp client example 20
Troubleshooting Case Mapper 23
Testing CaseMapper 23
References 24

Appendix: SME Parameters 25
Boolean Parameters 25
Numeric Parameters 26
Experimental Parameters 26

Overview

Case Mapper provides an interface for using computational models of structure-mapping theory [1]. The Structure-Mapping Engine (SME) [2,3] models analogical matching. MAC/FAC [4] models similarity-based retrieval. These are embedded in the FIRE reasoning engine [5], which provides a knowledge-base infrastructure and various reasoning facilities. Case Mapper is designed for cognitive scientists. This means we assume some familiarity with knowledge representation, predicate calculus and analogical reasoning, but we do not assume any programming skills on your part. Being a research software, Case Mapper is a bit fussier than a commercial application, but it is a software that our group uses daily in a variety of systems, hence the internal operations are quite robust.

This manual describes how to operate Case Mapper. Specifically,

- Use Knowledge Bases
    - Browse knowledge
    - Modify/Delete existing knowledge
    - Add new knowledge
- Use SME for performing analogical comparison.
- Use MAC/FAC for performing similarity based retrieval.
- Use KQML server for easy programmatic integration with external components.

For information about the algorithms behind these cognitive models and related experiments, please see the following references: [2,3,4]. These can be found on the QRG web site, at www.qrg.northwestern.edu. In learning Case Mapper, we suggest that you work through the examples in the order provided in the manual.

About Knowledge Bases and Cases

Case Mapper is always used with a knowledge base (KB). The KB stores knowledge, including facts and cases. Cases are simply collections of facts that can be treated as a unit. (Inside the KB, cases are implemented via Cyc-style microtheories. Sometimes browsing tools will refer to microtheories, and that's what they mean. You can just think of them as cases.) Some facts in the KB specify properties of the vocabulary of predicates it uses, including their *arity* (number of arguments), whether they are a relation or a logical function, and what types of arguments they can take. SME and MAC/FAC require this information for any predicate used in cases.

It is important to understand that there is a translation process that occurs when facts are added to an SME description. Recall that in structure-mapping theory, there is a sharp distinction between *attributes*, which are unary predicates on entities, and *relations*, which have two or more arguments. This is a psychologically important distinction. In Cyc-style knowledge bases, attributes are implemented via *collections*. Instances are related to collections via *isa* statements, i.e.,

```
(isa Nero Cat)
```

This says that Nero is a Cat, i.e., the individual Nero is an instance of the collection Cat. It is logically equivalent to the attribute form that SME expects, i.e.,

```
(Cat Nero)
```

Case Mapper automatically translates between these two forms as needed. When you see a case in the KB browser, you will see **isa** statements being used, whereas when you see the same case in the SME or MAC/FAC results browser, you will see it in the attribute form.

For more information on SME, including a discussion of its algorithm and more in-depth explanation of the terms discussed above, see [2,3].

The Case Mapper distribution comes with two knowledge bases:

- *SME Classics*: (`/smoke-kb/` directory[1]) This KB contains predicate specifications and cases from many of the early published SME and MAC/FAC experiments. It contains a minimal ontology, as well as all of the stimuli from the Karla the Hawk experiments and other examples from the literature. Experimenting with this KB is a great place to start.
- *OpenCyc-derived:* (`/opencyc-kb/` directory) This KB contains contents we extracted from the OpenCyc knowledge base. OpenCyc is a freely available[2] version of the Cyc knowledge base, which is currently the largest and most comprehensive formally represented knowledge base in the world. We do not use Cycorp's reasoning engine, since our FIRE reasoning engine is optimized for analogical operations. The subset of OpenCyc concepts we extracted currently contains just over 1.3M facts, 58K concepts, 14K relations, and 3K functions. We distribute this KB to support cognitive scientists who want to do larger-scale experiments.

You will find that some operations are much faster with the SME classics KB, although there is much more knowledge in the OpenCyc KB.

How to: Open a Knowledge Base

Once Case Mapper is open, your window should look similar to Figure 1.

1. Open the **KB** menu, and click **Open KB**.
2. Navigate to the directory where your KBs are stored (by default, it's **C:/QRG-KBs/CaseMapper/KBs** under Windows and for Linux it's **<CaseMapper>/Data/KBs**), and select a

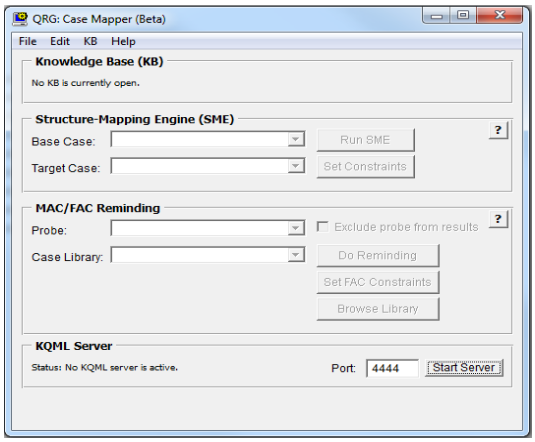


Figure 1: Case Mapper is open for business.

[1] It is sometimes called the smoke KB because it is automatically constructed every time the FIRE regression test is run. Regression tests are also known as smoke tests, by analogy with seeing if a new piece of equipment smokes when it is first plugged in.

[2] www.opencyc.org

KB by selecting the directory of the KB you wish to use (e.g. **smoke-kb** or **opencyc-kb**), as in Figure 2, and then click OK.

3. Case Mapper will display the current KB the Knowledge Base (KB) field.

Once a KB is open, you can browse and edit the KB (click **Browse KB** in the **KB** menu), add new knowledge to the KB with *meld files* (discussed below), or fetch cases for use with SME and MAC/FAC. To change KBs, click **Close KB** in the **KB** menu, and open another as described above.

How to: Back up a knowledge base

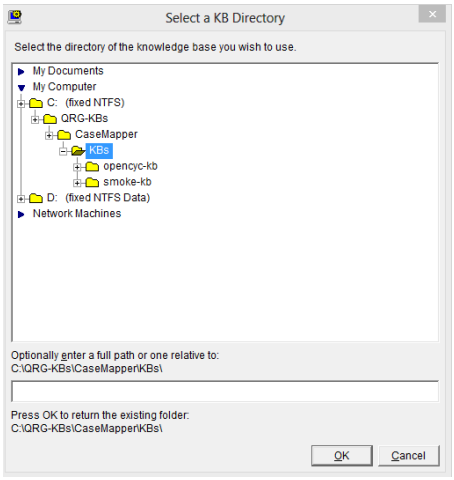


Figure 2: Opening a KB.

Things can go wrong with any program, and more so with research software. We strongly recommend backing up your KBs before doing anything else with Case Mapper. (In the worst case, you can always reinstall Case Mapper, but that takes even longer.) Once you open a KB (discussed next), you can back it up for later use. The command to use is on the KB menu, Create KB Backup. It will ask you for a directory to place the backup in. You can choose any directory you like (e.g. a USB flash drive), and the backup dialog lets you type in the name of a new subdirectory along that path which should be used. Importantly, *the **backup process deletes anything that is already in the directory** that you give it, to avoid KB corruption*. So unless you are replacing a previous backup, **creating a new directory** is almost always what you want to do. Should you need to use your backup, the command "Restore KB from Backup", also on the KB menu, can be used to replace the contents of a KB with the backup.

The SME Classics KB is tiny, requiring only around 10 MB of storage. The OpenCyc-derived KB requires less than 2GB of storage.

How to: Close Case Mapper

When you are through with Case Mapper, we recommend that you close the KB you were working with (i.e. KB | Close KB) and then exit the program in the usual way (File | Quit). We have set things up so that the KB should close automatically if you just quit, though.

How to: Browse a Knowledge Base

Case Mapper enables you to browse the knowledge base in a web browser[3]. After you open a KB, the menu command **Browse KB** in the **KB** menu will automatically open your browser to the correct address. Sometimes this process can be cranky, since communicating with browsers from other

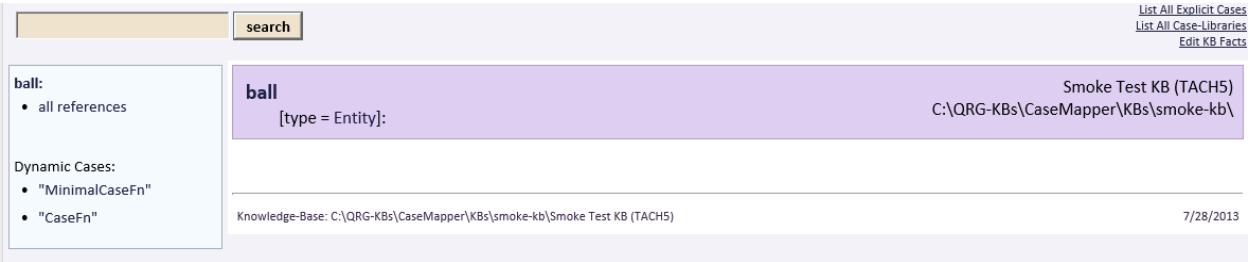
[3] We routinely use Internet Explorer, Chrome, and Firefox. We have not systematically tested our code with other web browsers.

software isn't always smooth. If the browser contents haven't changed within 30 seconds, press the button again, after starting a fresh copy of the web browser.

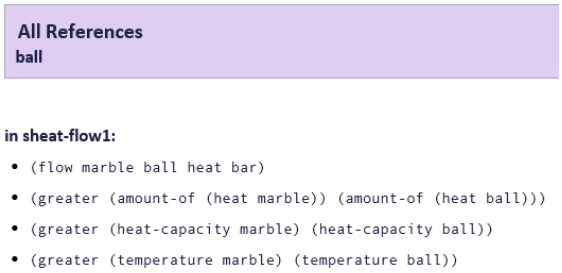
The browser interface allows you to search for terms in the KB and manually remove or *forget* facts from cases in the KB. Searching is easy: enter a search term, or the beginning of a search term (i.e. "ba" in the SME Classics KB will bring up concepts such as *bad, bad-tempered, baker,* etc.).

> **Warning**: Searching with a small search string (e.g. "ba") in a large KB (e.g. OpenCyc) on a slow computer (e.g. a laptop) may take a *very* long time to complete. It's better to search with a more specific term and use the links in the concept descriptions (discussed below) to navigate when possible.

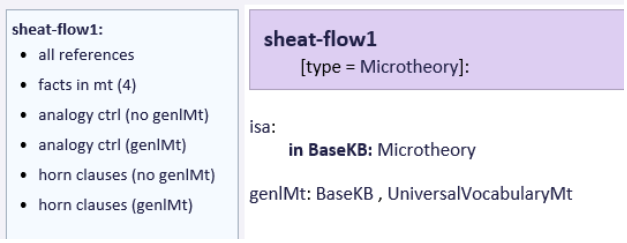
Let's pick *ball* from the list of concepts, using the SME Classics KB. What you'll see looks something like this:



Since ball is an entity, the set of operations we can do on it is limited. If you click on **all references**, you should see something like this:



This tells you that, in the SME Classics KB, the entity **ball** only appears in the case **sheat-flow1**. Cases in FIRE are implemented by what are called *microtheories* in the OpenCyc ontology. Thus if you click on the name of the case, you'll get something like this:



The **isa** statement is Cyc's way of doing attributes. The **isa:** can be read as, in the BaseKB microtheory, the statement (isa sheat-flow1 Microtheory) is true. When facts are translated into SME's format, such isa statements become attribute statements, e.g. (Microtheory sheat-flow1). Notice on the left that one can see the total size of the case (some of them are quite large). Analogy control predicates will be explained later.

On the right of every page is a short list of menu items. The most immediately useful one is List All Case-Libraries, which lists the case libraries that are currently in the knowledge base. You will see amongst them SyslitCaseLibrary, SyslitCaseLibrary12, and SyslitCaseLibrary5 (the others are regression tests for analogical generalization, not yet supported in Case Mapper). These are from the Karla the Hawk studies, and if you drill down into them, you will find a number of cases to experiment with.

To see the other capabilities of the KB Brower, please switch to the OpenCyc KB (by closing the SME Classics KB and then opening the OpenCyc KB). Search for "bas". If you click on the **BaseballDiamond** concept, you should see something similar to Figure 3. BaseballDiamond is a *Collection*, i.e., an attribute in structure-mapping terms, as noted in the header, and it's a subtype of (i.e., it *genls* to) **AthleticField**. You can click both Collection and AthleticField to learn more about both concepts – this is an efficient manner of navigating the knowledge-base.

To the left of the main frame, you may browse all the other *genls* and *specs* of the concept, as well as retrieve all references of the concept in the entire KB. Retrieving all references may take a very long time if there are a lot of them. For example, there are over a half-million `isa` statements in the KB, so retrieving all of them is rather time-consuming, and can even lead to time-out errors (see Troubleshooting).

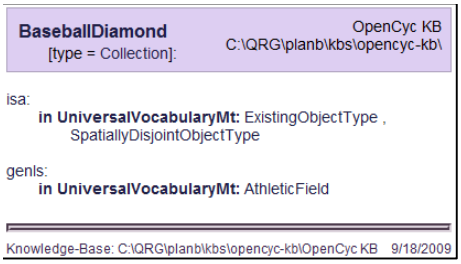


Figure 3: Browsing a KB concept.

To get a deeper understanding of CyCL (language used in Cyc), we recommend you to visit http://www.cyc.com/ and browse through the documentation.

How to: Fetch a Case

A case is a set of facts. Consequently, a case can represent an experience, a stimulus, a collection of domain knowledge, a theory, a story, or any other set of knowledge. Pre-existing cases may be fetched directly from your KB or new cases can be automatically constructed from portions of the knowledge

base (e.g., what the system knows about an entity, say a person named "Tom Hinrichs"). You can do so using an appropriate *case constructor*. The KB must be open before fetching cases.

To fetch an existing case from the KB:

1. Under the **File** menu, click **Fetch Case**, then **Fetch Existing Case**. A pop-up dialog will appear (Figure 4).
2. Locate the case by typing part of the case's name, and click **Search KB**.
3. Click the desired case and click **Load**.
4. The case name is now added to the *Base Case*, *Target Case*, and *Probe* drop-down lists in the main window.

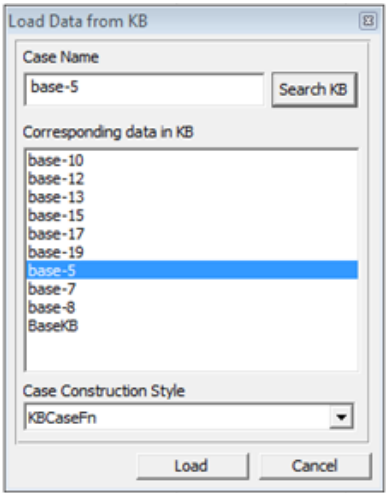


Figure 4: The case-loader popup dialog.

The example in Figure 4 is using the SME Classics KB, and the base stories are from the Karla the Hawk story set [4].

One can also dynamically generate cases from the contents of the knowledge base. To fetch an *entity* or *collection* case from the KB:

1. Under the **File** menu, click **Fetch Case**, then **Fetch Entity Case** or **Fetch Collection Case**. A pop-up dialog will appear (Figure 4).
2. Locate the entity or collection name in the KB by typing part of its name, and click **Search KB**.
3. Click the desired entity or collection.
4. Choose your desired case construction style (discussed later), and then click **Load**.
5. The case name is now added to the *Base Case*, *Target Case*, and *Probe* drop-down lists in the main window.

There are three case constructors that are built into Case Mapper currently. They are:

1. **MinimalCaseFn:** Retrieves the set of facts in the KB that mention the entity or collection.
2. **CaseFn:** Like MinimalCaseFn, except that it includes attribute information for every entity mentioned in the case.
3. **CleanCaseFn:** Like CaseFn, except that redundant attributes are removed.

FIRE supports a number of other case constructors (e.g. [6]), but these three are useful for many purposes.

After you've fetched cases, you can run SME. Alternatively, you can use the case as your probe for MAC/FAC. Both of these operations are discussed with examples below.

Using SME from within Case Mapper

We illustrate the use of SME via an example that uses the SME Classics ("smoke-kb") knowledge base. We will fetch two cases from the knowledge base and then browse an analogical mapping between the cases. These cases are from the "Karla the Hawk" stimulus set described in [3].

How to: Run SME

1. Open the "smoke-kb" knowledge base.
2. Fetch the case *base-10*.
3. Fetch the case *ta-10*.
4. Select **`(KBCaseFn base-10)`** from the **SME Base Case** dropdown list, and select **`(KBCaseFn ta-10)`** from the **SME Target Case** dropdown list. The SME frame of your window will resemble Figure 5.
5. Click the **Run SME** button.

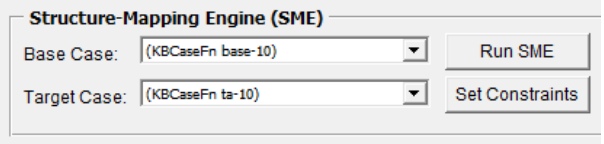


Figure 5: SME is ready to run.

If no message appears notifying you of a problem, SME has successfully computed analogical mappings between the base and target cases, and the mappings are ready to be browsed. If the web browser does not open automatically, press the **Run SME** button again.

How to: Browse SME mappings

The SME browser opens automatically after running SME. The browser interface provides an interactive report on the mappings that SME generated from the two representations. Browsing SME does not affect the mappings that were generated; this is just a manner of inspecting various details of SME's output. We'll start with the initial page, as shown in Figure 6, which results from following the instructions in the last section.

The SME number (2 in Figure 6) and the Mapping number (68 in Figure 6) will likely differ from your results; these are identifiers used to differentiate various SME instances and mappings in a particular session with Case Mapper. If you constrained SME (as we do in the next example), your constraints appear on this page as well.

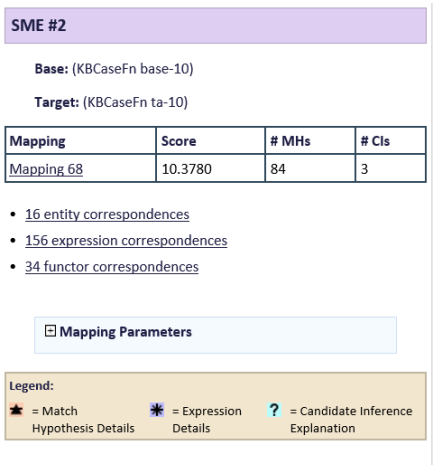


Figure 6: Browsing SME.

Clicking the Base or the Target case names will let you browse the facts in each representation. Clicking on any of the mappings will provide more information about that analogical mapping. Note that SME may compute multiple mappings, and in some cases, no mappings. These are not errors – the number and content of mappings is a function of the facts in the

base and target cases, constraints placed on the mapping process (discussed below) and the parameters of SME (discussed in the appendix).

Click on the mapping to browse it in more detail, as shown in Figure 7. At the top is a score (in this case, 10.378) that reports the overall *structural evaluation score* of the selected mapping. Note that this score is not normalized. You will also notice a list of *entity correspondences* from the base and target. Each correspondence has *support*, which are expressions that provide evidence for the correspondence, as well as a *match hypothesis score*, computed from the support, which reflects SME's confidence of the correspondence.

Candidate inferences are facts that may be true about the target, based on the mapping between base and target. Clicking the **candidate inferences** link will list the candidate inferences for this mapping.

The base-10/ta-10 candidate inferences are facts about temporal order of events and causality. You can browse each candidate inference by clicking the ? button – this will list the source and structural support for the inference. You will notice in this example expressions like (:skolem asset2). These are *analogy skolems*, entities whose existence is hypothesized by the candidate inference. Notice that SME is very conservative by default concerning candidate inferences, only inferring structure when supported by relational matches. For some purposes, e.g. building up a model of a new domain where little is known about the target, a more liberal criterion can be used by modifying the 'Allow entity-supported-inferences' SME parameter. SME parameters are discussed in appendix.

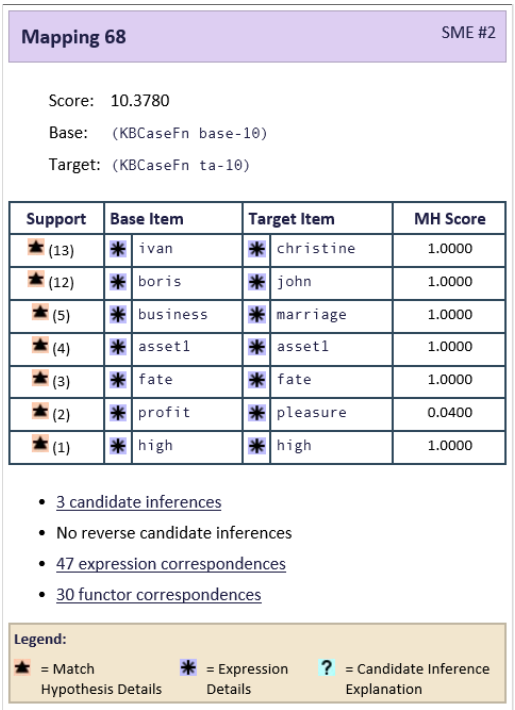


Figure 7: Browsing a SME mapping.

Expression correspondences are facts that correspond between the base and target cases. You can browse these by clicking the **expression correspondences** link while browsing a mapping. These expression correspondences are what suggest the entity correspondences. Moreover, since score "trickles down" to the arguments of a correspondence, the expression correspondences provide support for the entity correspondences, such that those entities which participate in large relational structures get higher scores.

How to: Constrain SME

By default, SME will produce the best structural mapping (or up to three, if the others are very close) for the given base and target. However, sometimes the context for analogical reasoning provides more constraints that need to be respected. In understanding an explanatory analogy, for example, the context provides explicit constraints that some entities must match (e.g., "An atom is like the solar system" means that atom and solar system must correspond). In analyzing a complex analogy, several

iterations of mapping can be required (e.g. [7]), so SME must be told to avoid certain correspondences, to force it to look for alternatives. Case Mapper provides an interface for adding and removing such constraints. To illustrate, let us consider a variation of the mapping we just used:

1. Fetch cases base-10 and ta-10 from the SME Classics KB.
2. Select **`(KBCaseFn base-10)`** as the SME base, and select **`(KBCaseFn ta-10)`** as the SME target. The SME frame of your window will resemble Figure 5.
3. Click **Set Constraints**.
4. Select **Ivan** from the entities in the base, and select **Christine** from the entities in the target.
5. Select "Selected entities must NOT correspond" from the **Available constraints** dropdown list, and click **Add**. The constraint popup should look like Figure 8.
6. Click **Done**, then click **Run SME** and browse the resulting SME mappings.

The SME browser will display the constraint we just added (in addition to any other constraints you've added). In this case, the mapping suffered a score decrease due to the constraint. If you browse the mapping, you'll notice that Ivan and Christine are not included in the entity correspondences, which also prevents many of the expression correspondences from participating in the mapping.

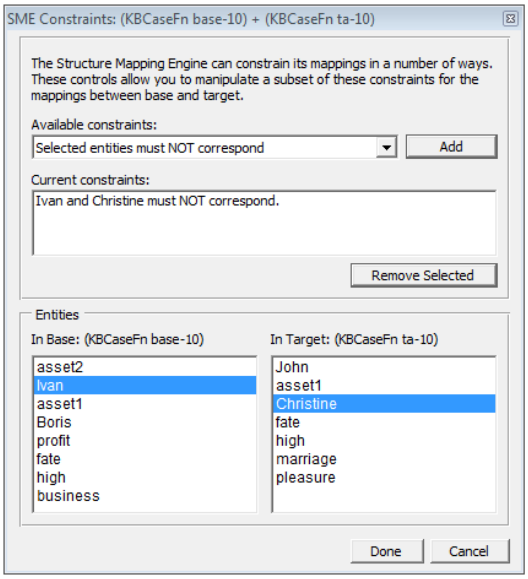


Figure 8: Constraining SME.

Using MAC/FAC from within Case Mapper

MAC/FAC is a model of similarity-based retrieval. The inputs to MAC/FAC are (1) a *probe*, which can be any existing or dynamically-created case, and (2) a *case library*, which is a set of persistent cases in the KB. You can use Case Mapper to select a probe and a case library, and run MAC/FAC transparently. You can select a probe by fetching a case from the KB, as discussed above. In the following, we discuss the processes of loading a case library and running MAC/FAC.

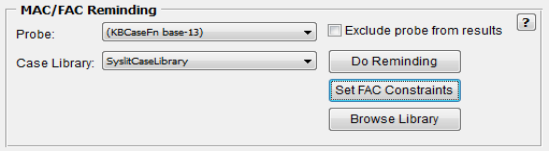


Figure 9: MAC/FAC in Case Mapper

How to: Select a Case Library

A case library is a set of cases. A single KB can contain many case libraries. The KB must be open before selecting a case libraries. To select a case library from the KB:

1. Under the **File** menu, click **Fetch Case**, then **Fetch Case Library**. A pop-up dialog will appear (Figure 10).
2. Locate the case library by typing part of the case library's name, and click **Search KB**. Note: If you are using the smoke-kb, there are several pre-existing case libraries, as in Figure 10.
3. Click the desired case library and click **Load**.

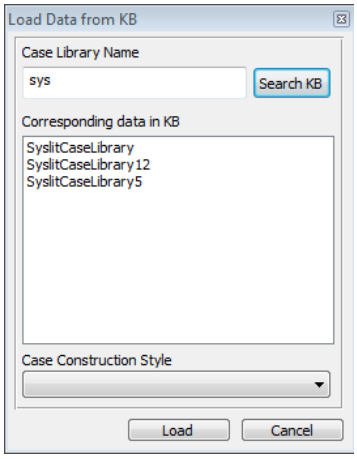


Figure 10: Case Library fetch dialog.

The selected case library will now be present in the Case Library drop-down list, within the MAC/FAC interface. For example, as shown in Figure 9 we have chosen ***(KBCaseFn base-13)*** as the probe and ***SysLitCaseLibrary*** as the case library. You can browse the given case library by clicking "Browse Library" (Figure 9). This will open up the KB browser and display the contents of the selected case library.

How to: Run MAC/FAC

Once you have loaded and selected a probe and a case library (discussed above):

1. Run MAC/FAC by clicking "**Do Reminding**" (Figure 9). If you select "Exclude probe from results," MAC/FAC will exclude the probe from the case library when performing similarity-based reminding. This is a good practice – after all, the most similar case to a probe is the probe itself.

2. Clicking “Do Reminding” will open a new browser window, with a display similar to Figure 11.
3. The browser window (Figure 11) has two frames, the lower one shows the results of MAC /FAC . The results of the MAC stage are shown as ‘MAC Results’, listing the cases with the highest content vector dot-product to the probe. These cases are inputs to the FAC stage, which uses SME to find the most similar case(s). The output(s) of the FAC stage are listed under “MAC/FAC Results”.
4. The upper frame (Figure 11) shows the browser interface for MAC/FAC which allows you to perform additional operations and also to run MAC/FAC again if necessary. Using this interface you can:
    a. Select a case from the Case Library list and click **SME-Compare to Probe**. This will open a SME mapping browser (discussed above), between the probe and the selected case.
    b. Select a case from the Case Library list and click **CV-Compare to Probe**. This will display the content vectors of the probe and the selected case, as well as the dot-product of the two vectors. This dot product is used in the MAC stage of MAC/FAC.
    c. Click **Run MAC/FAC** to run MAC/FAC again and see the results.

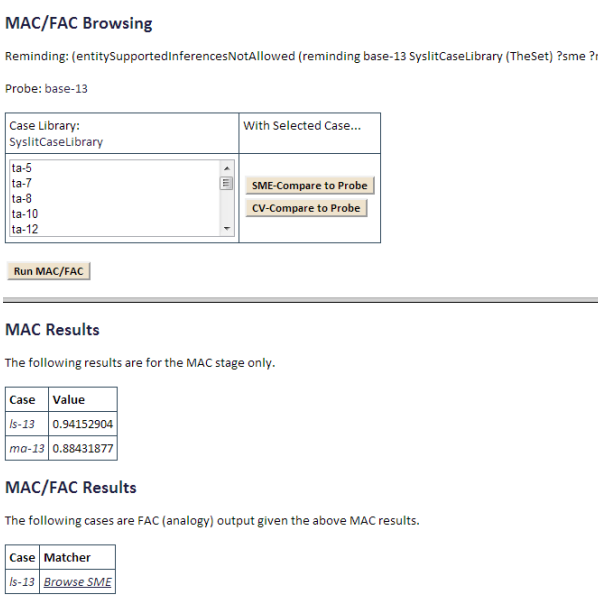


Figure 11: Viewing MAC/FAC results.

5. The FAC stage of MAC/FAC is achieved via SME comparisons between the probe and the candidates chosen in MAC stage. You can impose constraints on the comparisons using “**Set FAC Constraints**” (Figure 12). The interface is similar to “Set Constraints” (discussed before) except that we can add constraints related to only the entities in the probe.

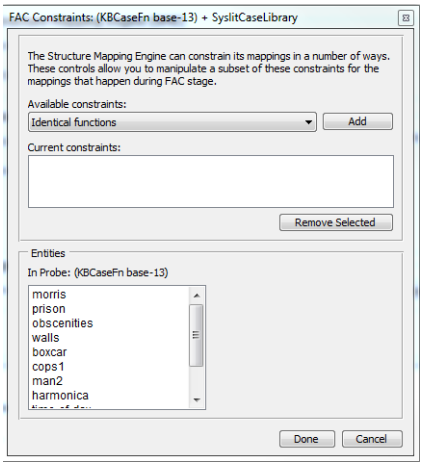


Figure 12: FAC Constraints

Extending a knowledge base

For your own experiments, you may need to write your own cases, including defining new predicates, relations, and attributes. This section explains how to do this.

How to: Write a case file

The file format supported by Case Mapper is the *meld* file (all files end in the extension .meld). You can see some examples of meld files by looking in the samples folder that was included in your installation of Case Mapper. Meld files are easy to write, with only a few simple rules:

1. Each case must start with a **case statement** of the form (Case *casename*) where *casename* is the name used to identify your case. (It is wise to use the KB browser to check and see if the name you are thinking of is already in use. Loading a meld file that refers to an existing case will add to, not replace, the facts involving that case.)
2. Facts following a case statement are stored within that case.
3. You can define multiple cases in one file, as long as each set of facts is set off by a separate case statement ,as Figure 13 illustrates.

```
(Case Bob1)
(isa Bob Dog)
(relationExistsExists loves Person Dog)
(animalTypeMakesSoundType Dog BarkingSound)
(relationInstanceExists anatomicalParts (GenericInstanceFn Dog) Mouth)

(Case Bob2)
(isa Bobby Cat)
(relationExistsExists loves Person Cat)
(relationInstanceExists anatomicalParts (GenericInstanceFn Cat) Mouth)
```

Figure 13. A simple meld file containing two cases: *Bob1* and *Bob2*. Each case contains a list of facts.

You can write a case file using any text editor that you are comfortable with (just make sure to save your file as *filename*.**meld.** Three important points:

1. It is important that your text editor produce plain text files (often called ASCII files or ".txt" files, in Save As menus), since formatted files (like .doc, .html, or .pdf) will not be correctly handled by Case Mapper's load routines.
2. Your case must use concepts and relations that are defined in the knowledge base for SME and MAC/FAC to work properly. You can find appropriate pre-existing concepts by using the KB browser. If you need to define your own, please see the next section.
3. To make editing a little easier, you might want to use a text editor that will match your parentheses for you (this means making sure that there are equal numbers of right and left

parentheses). If your file has mismatched parentheses it will not load in Case Mapper. We recommend Emacs, a freely-available text editor[4].

How to: *Define new predicates and concepts*

To define a new predicate, you must specify several things via facts in meld files:

1. Is it a relation or a function? If it is a relation, then you should include the statement
   `(isa <new> Relation)`
   in your file. If it is a logical function, then include
   `(isa <new> Function-Denotational)`
   The collections `Relation` and `Function-Denotational` are used by the KB/SME interface to determine whether something is a relation or a function. The minimum additional information that SME needs is its arity (number of arguments). For fixed-arity relations or functions,
   `(arity <new> <integer>)`
   where *<integer>* is the arity suffices. For variable-arity relations, state
   `(isa <new> VariableArityRelation)`
   and for variable-arity functions, include
   `(isa <new> VariableArityFunction)`
2. Is it an attribute? Attributes are implemented as collections (i.e., concepts), so you should include the statement
   `(isa <new> Collection)`
   in your file. You should also incorporate your new concept into the existing ontology, to help keep things conceptually clear. (You may find out in the process of doing so that a reasonable concept already exists, and hence you don't have to define a redundant one.) To do this, find a relevant concept that is a superclass of your concept, let's say *<super>*, and include in your file
   `(genls <new> <super>)`
   You can have multiple genls statements concerning a concept in your file, of course.
3. It is very useful to write comments about your new predicate. You do this by the following statement
   (comment <new> <string describing new>)
   in your file. These assertions will be used by the KB Browser, so that you (and potentially others) can be reminded about your intent. While SME will work without comments, we have learned over the years that this is a very useful practice.

In Cyc-derived KBs, the convention is that relations start with lower-case letters and are camel-cased (e.g., `connectedTo`), functions start with an upper-case letter and have "Fn" on the end (e.g., `FruitFn`), and attributes start with an upper-case letter (e.g., `BiologicalSpecies`). In the SME Classics KB, predicates, functions, and attributes are all lower-cased, with hyphens used to split terms instead of camel-casing. Case Mapper does not enforce these conventions, but you will find it useful to stick with them when possible.

[4] You can learn more about emacs online at http://www.gnu.org/software/emacs/.

Generally you will find it useful to create a new meld file with the specifications for your new predicates and attributes, separate from the meld files for whatever cases use them. To indicate that these are background facts, include at the start of the file the declaration

```
(in-microtheory UniversalVocabularyMt)
```

This will make your new information accessible from every case.

How to: Load a case file into your knowledge base

If you have knowledge in a meld file that you want to use with SME or MAC/FAC, you must load it into your KB. To load a meld file into your KB, first ensure that you have a KB open and then choose **Load File** → **Import .meld file** from the **File** menu.

How to: Test a case file

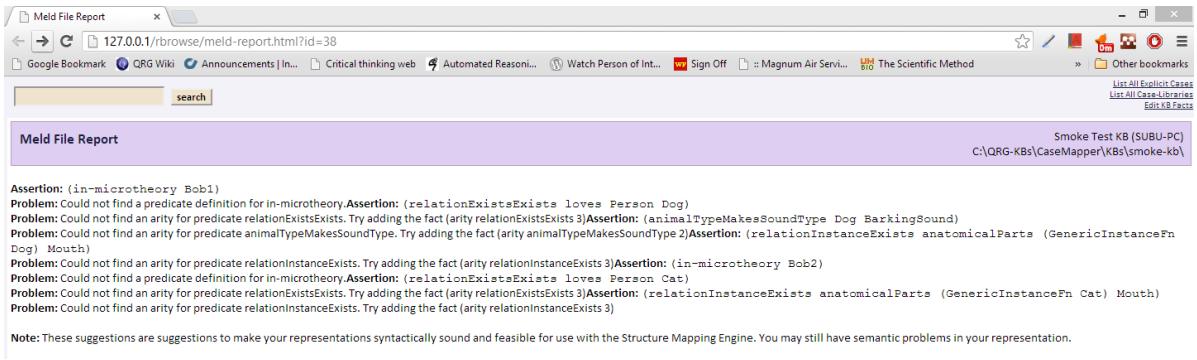

Meld File Report
127.0.0.1/rbrowse/meld-report.html?id=38
search
Meld File Report
Smoke Test KB (SUBU-PC)
C:\QRG-KBs\CaseMapper\KBs\smoke-kb\
Assertion: (in-microtheory Bob1)
Problem: Could not find a predicate definition for in-microtheory.Assertion: (relationExistsExists loves Person Dog)
Problem: Could not find an arity for predicate relationExistsExists. Try adding the fact (arity relationExistsExists 3)Assertion: (animalTypeMakesSoundType Dog BarkingSound)
Problem: Could not find an arity for predicate animalTypeMakesSoundType. Try adding the fact (arity animalTypeMakesSoundType 2)Assertion: (relationInstanceExists anatomicalParts (GenericInstanceFn Dog) Mouth)
Problem: Could not find an arity for predicate relationInstanceExists. Try adding the fact (arity relationInstanceExists 3)Assertion: (in-microtheory Bob2)
Problem: Could not find a predicate definition for in-microtheory.Assertion: (relationExistsExists loves Person Cat)
Problem: Could not find an arity for predicate relationExistsExists. Try adding the fact (arity relationExistsExists 3)Assertion: (relationInstanceExists anatomicalParts (GenericInstanceFn Cat) Mouth)
Problem: Could not find an arity for predicate relationInstanceExists. Try adding the fact (arity relationInstanceExists 3)
Note: These suggestions are suggestions to make your representations syntactically sound and feasible for use with the Structure Mapping Engine. You may still have semantic problems in your representation.

Figure 14: Sample Report from check meld

You can test the syntax of your case file by choosing File → **Check .meld file** from the **File** menu. This will produce a report page in your browser as shown Figure 14. For our example file in Figure 13, we have not defined the predicates 'relationExistsExists' and 'relationInstanceExists'.

KQML Server

Case Mapper comes with a built-in server module, so that other applications and cognitive models can access SME, MAC/FAC, and knowledge base programmatically. Also included is an example client application in the file **test-client.lsp**. This is a Common Lisp implementation of a stand-alone program that sends and receives *KQML* (Knowledge Query Manipulation Language) packets to the Case Mapper server. Importantly, the KQML server interface of Case Mapper can communicate with client programs written in any programming language, provided they can open sockets and send properly-formatted KQML messages. KQML-based interaction with Case Mapper involves four types of interactions: (1) Knowledge Base *store*; (2) Knowledge Base *retrieve*; (3) Knowledge Base *forget*; and (4) *ask*. Next, we discuss how to start and stop the KQML server, and we review KQML packet structure and specifics including each of these four interactions.

How to: Start & Stop the KQML Server

On the bottom of the Case Mapper interface is a **KQML Server** panel. This panel shows the status of the server, and contains *Port* field for entering the server port of your server (default is 4444), and a button for starting and stopping the server. The status label will change based on the status of the server.

Once you have started a server, you can use a client program written in the programming language of your choice to communicate with Case Mapper and use the knowledge base, the built-in reasoner, and SME.

Basic KQML & Packet Structure

Client programs can be written in any language, permitted that they can open a TCP/IP socket to the computer running Case Mapper. Case Mapper allows a subset the standard KQML interactions, so we can review them in their entirety. The following string is a valid KQML packet that sends the content `(ask :query (isa ?x Dog))` to Case Mapper.

```
(:tell :sender "localhost"
       :receiver "localhost"
       :in-reply-to ""
       :reply-with "MSG #2"
       :language ""
       :ontology ""
       :content (ask :query (isa ?x Dog)))
```

Note that all of the keyword fields, except for `:content`, are specified by character strings, and the packet must contain quote (") characters around these parameters. Conveniently, you can use the above packet structure and substitute only the parameters for `:reply-with` and `:content` to programmatically access Case Mapper.

The Case Mapper server accepts these packets in string format, and its response packets are in the same format. Note that the Case Mapper server socket connection *is not* persistent with its clients, so it disconnects each time a packet is processed, after a response packet is sent back to the sender. In the following subsections, we describe each interaction and packet. Note that only the `:content` field changes for each of the interactions.

How to: Use KQML to Ask

The above packet example with content `(ask :query (isa ?x Dog))` performs an *ask* operation on the Case Mapper server with the query . Consequently, Case Mapper will use its working memory and its KB to find all possible answers to the asked query – in this case, it will bind the variable `?x` to all known instances of the collection `Dog`. The response will look something like this:

```
(:tell :sender "localhost"
```

```
:receiver "localhost"
:in-reply-to "MSG #2"
:reply-with "MSG #2"
:language ""
:ontology ""
:content ((ist-Information EverythingPSC (isa Bluto Dog))
          (ist-Information EverythingPSC (isa Dash-VictoriasPet Dog))
          (ist-Information EverythingPSC (isa HYP-Dog-5968235 Dog))
          (ist-Information EverythingPSC (isa (GenericInstanceFn Dog) Dog))))
```

Statements of the form

`(ist-Information <mt> <fact>)`

Indicate that fact *<fact>* holds in microtheory *<mt>*. Typically you will be doing queries with regard to your own cases. `EverythingPSC` is a problem-solving context built into the OpenCyc ontology that treats the entire KB as one massive context. This is good for spelunking through the KB, but almost always is a bad idea otherwise.

The SME and MAC/FAC functions are accessible programmatically via *ask* messages. We discuss this in further detail below.

***How to**: Use KQML to Store*

Storing is the act of inserting new facts into Case Mapper's KB. You can create a store message by replacing the `:content` field of the above packet with `(store :fact <fact> :context <context>)`. The context is a required argument for specifying the logical context in the KB inside which the fact will be stored. Contextualizing knowledge in the KB is important for maintaining the integrity of cases and for enabling reasoning and learning about rich human knowledge (e.g. FrodoBaggins is a Hobbit, in the MiddleEarthMt).

***How to**: Use KQML to Retrieve*

Retrieving is the act of finding facts within Case Mapper's KB. You can create a retrieve message by replacing the `:content` field of the above packet with `(retrieve :pattern <query> :context <context>)`. The context is an optional argument for specifying the logical KB context to perform the retrieve.

***How to**: Use KQML to Forget*

Forgetting is the act of removing a fact from a logical context in Case Mapper's KB. Because knowledge is contextualized, forgetting a fact in one context will not affect the presence of a fact in another context. You can create a forget message by replacing the `:content` field of the above packet with `(forget :fact <fact> :context <context>)`. The context is a required argument for specifying

the logical context in the KB inside which the fact will be forgotten. If the fact is not present in the given context, no action will be taken.

Using the test-client.lsp client example

This section assumes you have a Common Lisp IDE installed. Also note that this section uses certain cases within the "smoke-kb," but the same methods apply for any case in any Case Mapper-compatible knowledge base.

1. Start Case Mapper, open the "smoke-kb," and start your KQML server on some port.
2. Compile and load the test-client.lsp file.
3. In your IDE, evaluate: `(make-test-client "localhost" 4444)`, but substitute 4444 for the port you chose. This creates and binds a persistent client object for this server/port combination.
4. In your IDE, evaluate: `(send-test-msg '(ask :query (isa ?x harmful)))`. This splices the lisp form `(ask :query (isa ?x harmful))` into the `:content` argument of a new KQML message and sends it to Case Mapper. The outgoing message and response message should be printed in the IDE window, and the function should return the content from the response message, which contains Case Mapper's response to the *ask* query. Note that the term ?x is considered a *variable* by Case Mapper's internal reasoner, since it is proceeded by a question mark. As a context is not specified EverythingPSC (all of KB) is assumed to be the context.
5. In your IDE, evaluate the following: `(send-test-msg '(store :fact (isa America country) :context cm-test-case))`. This stores the fact `(isa America country)` in the KB, in a microtheory (contextual case) called `cm-test-case`. If the context `cm-test-case` did not exist beforehand, Case Mapper creates it.
6. Verify that the fact was properly stored by browsing the KB (via the KB menu) and searching for cm-test-case. You'll notice that there is one fact in the microtheory – the one we just stored programmatically using our KQML client.
7. *Retrieve* the case by evaluating: `(send-test-msg '(retrieve :pattern (isa America country) :context cm-test-case))` (note that the :context keyword is optional here), and *forget* the fact by evaluating: `(send-test-msg '(forget :fact (isa America country) :context cm-test-case))`. Note that you need not retrieve facts before forgetting them; we did this purely for demonstrative purposes.
8. Verify that the fact was properly deleted from the KB by refreshing your KB browser window.

Next, we discuss how to run SME using KQML *ask* messages.

1. If you are starting a new Case Mapper or Lisp IDE session, complete steps 1-3 above.
2. In your IDE, evaluate:
```
(send-test-msg '(ask :query (matchBetween
                             (ExplicitCaseFn solar-system)
                             (ExplicitCaseFn rutherford-atom)
                             (TheSet) ?match)
                             :context cm-test-context))
```
   The predicate *matchBetween* invokes SME, and the predicate *TheSet* indicates that its arguments should be viewed as a set. Here, since there are no constraints, (TheSet) is indicating the empty set. The ':context *cm-test-context*' indicates that the mapping will be performed with respect to cm-test-context. This allows the user to customize which predicates should be considered for analogy using analogy-control-predicates.
3. Case Mapper's response message should include the content:

```
((ist-Information cm-test-context
  (matchBetween (ExplicitCaseFn solar-system)
                (ExplicitCaseFn rutherford-atom)
                (TheSet) (MatcherFn 7 0))))
```

   Note: your `(MatcherFn 7 0)` term may differ in numerical arguments; this is only an internal identifier token that serves as a handle for getting further information about this SME operation, while it is still resident in Case Mapper's working memory[5].
4. Using the MatcherFn term you received back from Case Mapper (here, `(MatcherFn 7 0)`), you can perform the following *ask* queries to obtain data about this SME analogy operation. Just put the following lisp-forms into the :query argument of send-test-msg:

Match-level queries:

`(numberOfMappings` **<matcher-handle>** `?x)`

…returns the total number of mappings computed (here, ?x = 2).

`(bestMapping` **<matcher-handle>** `?x)`

…returns a handle to the best mapping (here, ?x = (`MappingFn 9 (MatcherFn 7 0)`)).

Once you have a handle to a mapping, you can use the following mapping-level queries:

Mapping-level queries:

`(numberOfCorrespondences` **<mapping-handle>** `?x)`

…returns the total number of correspondences in the mapping (here, ?x = 11).

[5] The first integer refers to the order in which an SME was created during a session, and the second integer refers to the number of times its base or target has been extended, since SME supports incremental mapping. In this example, this is the 7th SME created during the session, and no additional facts have been added to the base and target since their creation.

`(numberOfCandidateInferences` **<mapping-handle>** `?x)`

…returns the total number of candidate inferences in the mapping (here, ?x = 3).

`(structuralEvaluationScoreOf` **<mapping-handle>** `?x)`

…returns the structural evaluation score of the mapping (here, ?x = 0.1657).

`(correspondsInMapping` **<mapping-handle>** `?x ?y)`

…returns a list of ?x (base) and ?y (target) correspondences in the given mapping, for example: `(correspondsInMapping (MappingFn 9 (MatcherFn 7 0)) planet electron)`

`(hasCorrespondence` **<mapping-handle>** `?x)`

…returns a list of *correspondence handles* for all correspondences in the mapping, for example: `(MhFn 1 (MatcherFn 7 0))`. This is in the form `(MhFn` **<index>** **<matcher-handle>**`)`, where Mh stands for Match Hypothesis. Once you have correspondence handles, you can ask correspondence-level queries. Similarly, the next two queries produces correspondence handles.

`(correspondenceForBaseItem` **<mapping-handle>** `planet ?y)`

…returns a list of the form `(correspondenceForBaseItem` **<mapping-handle>** `planet` **<correspondence- handle>**`)`.

`(correspondenceForTargetItem` **<mapping-handle>** `electron ?y)`

…returns `(correspondenceForTargetItem` **<mapping-handle>** `electron` **<correspondence-handle>**`)`.

`(candidateInferenceOf ?x` **<mapping-handle>**`)`

…returns a list of `(candidateInferenceOf` **<ci-handle>** **<mapping-handle>**`)`, items, where each **<ci-handle>** can be used in the below Candidate inference-level queries. These are analogical inferences from the base case to the target case.

`(reverseCandidateInferenceOf ?x` **<mapping-handle>**`)`

…returns a list of `(reverseCandidateInferenceOf` **<ci-handle>** **<mapping-handle>**`)`, items, where each **<ci-handle>** can be used in the below candidate inference-level queries. These are analogical inferences from the target case to the base case.

Correspondence-level queries:

`(correspondenceBaseItem` **<correspondence-handle>** `?x)`

…returns the base item of the given **<correspondence-handle>**.

`(correspondenceTargetItem` **<correspondence-handle>** `?x)`

…returns the target item of the given **<correspondence-handle>**.

Candidate inference-level queries:

`(candidateInferenceContent` **<ci-handle>** `?x)`

…returns the fact-form of the given **<ci-handle>** in the mapping.

`(supportScoreOf` **<ci-handle>** `?x)`
…returns the *support score* of the corresponding candidate inference.

`(extrapolationScoreOf` **<ci-handle>** `?x)`
…returns the *extrapolation score* of the corresponding candidate inference.

`(candidateInferenceCorrespondences` **<ci-handle>** `?x)`
…returns a set of correspondence handles representing the correspondences (match hypotheses) that support the given candidate inference.

We have demonstrated how to use SME via Case Mapper's KQML server with a Common Lisp client; however, all of these queries can be performed by other programs written in other languages, provided they use the same KQML message structure and query syntax as discussed above.

Troubleshooting Case Mapper

- When you start the KB Browser or try to browse SME or MAC/FAC results for the first time, you may get a Windows Firewall pop-up dialog that asks you whether or not you want to allow Case Mapper to communicate. It is important that you choose "unblock", since otherwise the program will not be able to communicate with the browser.
- If you get a dialog box that says "unrecoverable error" when opening a KB, it means that either there isn't a KB at the location you selected or it is corrupted. Delete whatever files are at that location, and restore your KB from the backup you made.
- If you are browsing and you get something that says "Lisp error" instead of what you expected, then you hit a time-out in the browsing system. Either the operation is just too big for the browsing system to handle (e.g., showing all `isa` statements in the OpenCyc-derived KB) or your machine was spending a lot of time elsewhere. In the latter case, try again.
- When you get an unrecoverable error dialog, if you click on "debug" instead of "ok", the program will in some circumstances keep going just fine. However, you should save your KB soon, and be sure to have a backup handy just in case.
- If you have difficulty starting or connecting to the Case Mapper KQML server, ensure that your Windows firewall is disabled, at least for the port you have specified as your KQML server port.

Testing CaseMapper

You can test the basic functions of CaseMapper using **'Help> Test CaseMapper'** menu option.

- This operation requires you to have 'smoke-kb' as the current KB. So close any other KB and open smoke-kb (included with the installation).
- A file selection dialog will pop up. This lets you specify the name and location for the test result file.
- The result file will contain the current SME-parameter settings and other test statistics.

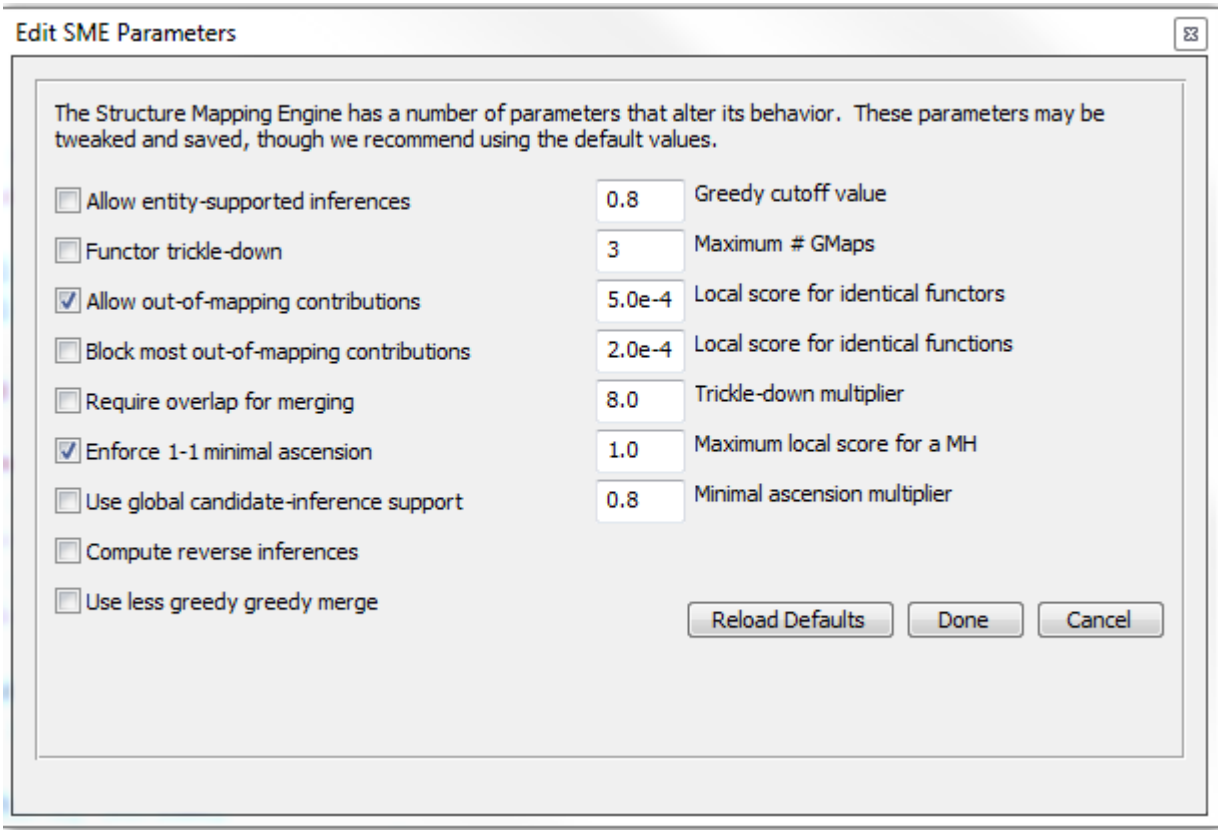
We request you to include the result file along with any bug report.

References

1. Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, **7**, 155-170.
2. Falkenhainer, B., Forbus, K. and Gentner, D. (1989). The Structure Mapping Engine: Algorithm and examples. *Artificial Intelligence*, **41**, 1-63.
3. Forbus, K., Ferguson, R. and Gentner, D. 1994. Incremental structure-mapping. *Proceedings of the Cognitive Science Society*, August.
4. Forbus, K., Gentner, D., and Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, **19**, 141-205.
5. Forbus, K., Hinrichs, T., de Kleer, Usher, J., J., Klenk, M., Lovett, A., and Paritosh, P. (unpublished) The FIRE Manual.
6. Mostek, T., Forbus, K, and Meverden, C. (2000). Dynamic case creation and expansion for analogical reasoning. *Proceedings of AAAI-2000*. Austin, TX.
7. Falkenhainer, B. (1987). An examination of the third stage in the analogy process: Verification-based analogical learning. In *Proceedings of IJCAI-87*. Los Altos: Morgan-Kaufmann.

Appendix: SME Parameters

CaseMapper provides an interface (**Edit | SME Parameters**) for customizing SME by modifying the default parameters. We recommend that you become very familiar with the SME algorithm before changing any of the default settings.



The descriptions of these parameters are given below.

Boolean Parameters

- **Allow entity-supported inferences:** We believe that there are two criteria people use for generating candidate inferences. The default criterion is conservative, only producing candidate inferences when supported by relational correspondences. However, there is another criterion that we believe is used when little is known about the target, i.e. allowing candidate inferences based on correspondences just involving entities. When this Boolean flag is false, SME uses the default conservative criterion. When it is true, it also allows candidate inferences to be generated using the more risky criterion. In Case Mapper this flag defaults to false, but in learning systems built on top of SME, it is often set to true.
- **Allow out-of-mapping contributions:** This determines how the final mapping score is computed. The default is set to true, indicating that the mapping score is the sum of all MHs in the mapping.
- **Block most out-of-mapping contributions:** The default is false. When set to true (checked), this limits the initial MH score trickle-down to mostly structurally consistent parents.

- **Enforce 1-1 minimal ascension:** The default is true. This allows functor correspondences to be created for relations matched via minimal ascension. This in turn means that non-identical predicate matches must be identical across the entire mapping.
- **Compute reverse inferences:** This allows SME to compute candidate inferences from the target to the base as well as from the base to the target. This is very useful in computing alignable differences.

Numeric Parameters

- **Maximum # Gmaps:** The maximum number of mappings to compute. The default value is 3.
- **Greedy cutoff value:** If there is a steep drop-off in score from the best mapping, lower-scoring mappings are dropped. This parameter sets the threshold for the cutoff, i.e. subsequent mappings must be within this percentage of the best mapping. The default value is 0.8.
- **Local score for identical functors:** The local score for a match hypothesis between expressions which have identical functors. The default value is 0.0005.
- **Local score for identical function:** The local score for a match hypothesis which has identical functions. This is typically smaller because functions, representing dimensions, can easily match non-identically if they support a larger matching structure. The default value is 0.0002.
- **Trickle-down multiplier:** Systematicity is implemented via a trickle-down algorithm, where the score of a match hypothesis cascades down to the match hypotheses of its corresponding arguments. The default value is 8.0.
- **Maximum local score for a MH:** Provides the ceiling for match hypothesis scores. The default value is 1.
- **Minimal ascension multiplier:** Indicates the penalty to be imposed if the alignment is due to minimal ascension. The default is that only 80% of the score will be considered.

Experimental Parameters

- **Functor trickle down:** Allows trickle-down to match hypotheses connecting functors.
- **Require overlap for merging:** An experimental new version of greedy merge which requires kernels to overlap. Normally all that is required is that they are not structurally inconsistent.
- **Use global candidate inference support:** Use global mapping information when calculating candidate inference support scores. Normally scores are computed for the correspondences on which the inference rests locally.
- **Use less greedy greedy merge:** Use an experimental form of greedy merge allowing addition of consistent kernels from other mappings.